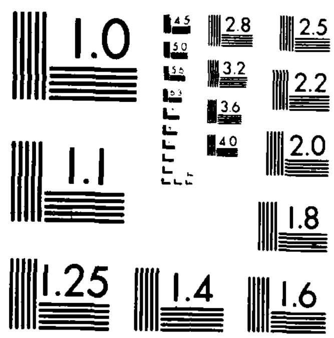
AD-A191 577 NARRATIVE COMPRESSION CODING FOR A CHANNEL WITH ERRORS 1/1
(U) NAVAL OCEAN SYSTEMS CENTER SAN DIEGO CA J W BOND
JAN 88

UNCLASSIFIED F/G 25/5 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

UNCLASSIFIED

DTIC FILE COPY

AD-A191 577

REPORT DOCUMENTATION PAGE

1b. RESTRICTIVE MARKINGS

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT

Approved for public release; distribution is unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION: Naval Ocean Systems Center

6b. OFFICE SYMBOL (if applicable): NOSC

7a. NAME OF MONITORING ORGANIZATION: Naval Ocean Systems Center

6c. ADDRESS (City, State and ZIP Code): San Diego, California 92152-5000

7b. ADDRESS (City, State and ZIP Code): San Diego, California 92152-5000

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (if applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	AGENCY ACCESSION NO.
Inhouse			

11. TITLE (include Security Classification)

Narrative Compression Coding for a Channel with Errors

12. PERSONAL AUTHOR(S)

J.W. Bond

13a. TYPE OF REPORT: Professional paper

13b. TIME COVERED: FROM Jun 1987 TO Jun 1987

14. DATE OF REPORT (Year, Month, Day): January 1988

15. PAGE COUNT

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

comma-free data
data compression capabilities
Huffman Code

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Data compression codes offer the possibility of improving the thruput of existing communication systems in the near term. This study was undertaken to determine if data compression codes could be utilized to provide message compression in a channel with up to a 0.10 bit error rate.

The data compression capabilities of codes were investigated by estimating the average number of bits-per-character required to transmit narrative files. The performance of the codes in a channel with errors (a noisy channel) was investigated in terms of the average numbers of characters-decoded-in-error-per-bit-error and of characters-printed-in-error-per-bit-error.

Results were obtained by encoding four narrative files, which were resident on an IBM-PC and use a 58 character set. The study focused on Huffman codes and suffix/prefix comma-free codes. Other data compression codes, in particular, block codes and some simple variants of block codes, are briefly discussed to place the study results in context.

Comma-free codes were found to have the most promising data compression because error propagation due to bit errors are limited to a few characters for these codes. A technique was found to identify a suffix/prefix comma-free code giving nearly the same data compression as a Huffman code with much less error propagation than the Huffman codes. Greater data compression can be achieved through the use of this comma-free code with code word assignments based on conditioned probabilities of character occurrence.

Presented at Agard Electromagnetic Wave Propagation Panel, Lisbon, Portugal, 19-23 October 1987.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT

☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL: J.W. Bond

22b. TELEPHONE (include Area Code): (619) 553-4166

22c. OFFICE SYMBOL: Code 83

DD FORM 1473, 84 JAN

83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

NARRATIVE COMPRESSION CODING FOR A CHANNEL WITH ERRORS

11 June 1987

DTIC COPY INSPECTED 1

Prepared by:

Naval Ocean Systems Center
Submarine Broadcast Systems Division
San Diego, CA 92152-5000

88 3 21 083

NARRATIVE COMPRESSION CODING FOR A CHANNEL WITH ERRORS

By Dr. James W. Bond, Staff Scientist
NAVOCEANSYSCEN, Code 83
271 Catalina Boulevard
San Diego, California 92152-5000

ABSTRACT

Data compression codes offer the possibility of improving the thruput of existing communication systems in the near term. This study was undertaken to determine if data compression codes could be utilized to provide message compression in a channel with up to a .10 bit error rate.

The data compression capabilities of codes were investigated by estimating the average number of bits-per-character required to transmit narrative files. The performance of the codes in a channel with errors (a noisy channel) was investigated in terms of the average numbers of characters-decoded-in-error-per-bit-error and of characters-printed-in-error-per-bit-error.

Results were obtained by encoding four narrative files, which were resident on an IBM PC and use a 58 character set. The study focused on Huffman codes and suffix/prefix comma-free codes. Other data compression codes, in particular, block codes and some simple variants of block codes, are briefly discussed to place the study results in context.

Comma-free codes were found to have the most promising data compression because error propagation due to bit errors are limited to a few characters for these codes. A technique was found to identify a suffix/prefix comma-free code giving nearly the same data compression as a Huffman code with much less error propagation than the Huffman codes. Greater data compression can be achieved through the use of this comma-free code with code word assignments based on conditioned probabilities of character occurrence.

INTRODUCTION

Data compression encoding offers an option for increasing the channel capacity of existing communications systems by efficiently encoding the narrative portions of messages. A data compression code assigns short binary code words to characters with a high frequency of occurrence and long code words to characters with a low frequency of occurrence. Difficulties arise when data compression codes are used in noisy channels because one bit error can lead to multiple character errors due to temporary loss of character synchronization.

This study focused on the investigation of Huffman and comma-free data compression codes which could be used to encode characters based on their probabilities of occurrence. The comma-free code results were then extended to encoding of characters based on their conditional probabilities of occurrence. In addition, several coding approaches using block codes and simple variants of block codes are discussed.

The data compression provided by a code is measured by the average number of bits-per-character of the encoded narratives files; the performance of the code in noisy channels is measured by the average number of characters-decoded-in-error-per-bit-error and the average number of characters-printed-in-error-per-bit-error. Generally speaking, as the number of bits-per-character decreases (that is, as data compression increases), the numbers of characters-decoded and printed-in-error-per-bit-error increase. Observe that under the assumptions of a fixed bit-error-rate and random bit errors, the ratio of the average number of character (decoded or printed) errors for two codes is equal to the ratio of the product of the average number of bits-per-character and the average number of characters (decoded or printed)-in-error-per-bit-error for the two codes.

Results are presented for a 58 character alphabet derived from the 95 character set of the personal computer and for processing narrative files stored on its hard disk. These files were edited to use only capital letters and certain seldom used symbols were deleted to obtain an alphabet emulating the Military Standard of the American Variation of the International Telegraph Alphabet No. 2 (hereafter called the Military Baudot Code) in use for Navy communications.

The error properties of both Huffman and comma-free codes depend on the specific choices of bits and code words, respectively, used to construct the codes. A main thrust of this paper is to identify the Huffman codes and the comma-free codes giving the lowest average number of character (decoded or printed)-errors-per-bit-error for a given compression gain.

APPROACH

Huffman codes are known to provide the best data compression possible for variable length coding of individual characters [reference 1]. This property is ensured by the code construction process because it is based on the probabilities of occurrences of the characters to be encoded. We began our investigation by establishing the properties of Huffman codes in noisy channels using character encoding.

The comma-free codes analyzed in this report are constructed using a sequential procedure found by R. A. Scholtz [references 2 and 3]. His procedure does not utilize probabilities of occurrence to guide the construction process. We developed a way to most nearly match the word lengths of a comma-free code to those of an optimum Huffman code in order to maximize the data compression performance of the selected comma-free code.

Even after specifying the distribution of word lengths of Huffman and comma-free codes, there are degrees of freedom in the construction processes. It was discovered that the error properties of a code depended on the use made of these degrees of freedom.

The insights provided by the investigation of Huffman codes and comma-free codes led to the identification of certain natural extensions of the presently used Military Baudot code. A comparison of the performance of these codes with those of Huffman and comma-free codes provides an additional performance gauge against which the latter codes can be assessed.

The Huffman code construction process has a great number of degrees of freedom. The impact of bit errors on character synchronization and character errors is very context-dependent; therefore, an analytical study of the dependency of error statistics on the Huffman construction process could not be performed. Therefore a simulation program was written and exercised to search for the best Huffman codes.

The number of degrees of freedom in the comma-free code construction process depends on the number of sequential steps in the process and not on the character set size. The performance of codes constructed in a few steps, which are the codes of most interest, are established analytically by an exhaustive treatment of the available codes.

Huffman and comma-free coding could also be applied to character encoding based on conditional probabilities of character occurrence. We obtained results for comma-free codes, which are placed in perspective by considering block coding of the words in large dictionaries.

DATABASE

Four narrative files were used to analyze the performance of data compression codes. These files (labeled I, II, III, and IV) contain 31,744, 28,672, 12,288, and 13,312 bytes, respectively, and are resident on the hard disk of the personal computer. All of the narrative files were technical documents involving some equations.

Table 1 presents the probabilities of occurrence for the different characters for each of the four narrative files. Note that the probabilities of occurrence of the characters are similar for the four narrative files.

BLOCK CODES AND GENERALIZED BAUDOT CODES

This section describes block codes and a family of compression codes which have a structure very similar to that of the presently used Military Baudot code.

Block Codes

A code consisting of code words of equal length is called a block code. The length of the block code used depends on the number of different symbols being encoded. A 58 character set is used by the United States Navy. Six-bit code words are necessary to encode this character set.

If words are encoded, then the dictionary size determines the length of the code words necessary to encode the words: 14 bits are necessary to encode a 16,384 word dictionary; 15 bits to encode a 32,768 word dictionary; and 16 bits to encode a 65,536 word dictionary. (Note for an average word length of five characters, the dictionary schemes use 2.33, 2.50, and 2.67 bits per character encoding, respectively, by not requiring the encoding of spaces.) Larger dictionaries appear to be prohibited by the difficulty of implementing encoding and decoding by table look-up operations. One final note: if block codes are used, a bit error leads to a single character error if characters are encoded or a single word error if words are encoded.

Since the English language is known to contain over 300,000 words some provision must be made to handle words not in the dictionary. It is undesirable to limit the vocabulary of the writer of Navy messages so that words occur which are not in the dictionary. In order to take the most advantage of a fixed-size dictionary some way should be found to handle variants of the same word, such as alternate spellings, misspellings, and abbreviations. We do not analyze block encoding of large dictionaries in this paper. Rather we chose to analyze the use of data compression coding of characters based on conditional probabilities of character occurrence.

Generalized Baudot Codes

The Military Baudot code consists of information carrying characters and two shift characters, which change the decoding of the next code word. The Military Baudot alphabet consists of 56 information characters and shift characters. One shift character shifts letters to figures and the other from figures back to numbers. If a simple block code was

TABLE 1. CHARACTER PROBABILITIES OF OCCURRENCE FOR FOUR NARRATIVE FILES

CHARACTER	NARRATIVE FILE I	NARRATIVE FILE II	NARRATIVE FILE III	NARRATIVE FILE IV
" "	0.3085	0.3171	0.2851	0.4095
E	0.0855	0.0865	0.0882	0.0664
T	0.0636	0.0677	0.0585	0.0492
N	0.0543	0.0537	0.0483	0.0407
O	0.0416	0.0518	0.0505	0.0420
I	0.0513	0.0511	0.0537	0.0426
A	0.0455	0.0450	0.0499	0.0434
R	0.0392	0.0421	0.0471	0.0438
S	0.0372	0.0391	0.0440	0.0367
H	0.0277	0.0263	0.0291	0.0162
C	0.0193	0.0232	0.0245	0.0216
L	0.0233	0.0218	0.0262	0.0226
D	0.0191	0.0206	0.0327	0.0165
U	0.0142	0.0185	0.0147	0.0165
P	0.0163	0.0170	0.0171	0.0164
M	0.0120	0.0162	0.0230	0.0150
F	0.0151	0.0141	0.0190	0.0169
G	0.0117	0.0121	0.0163	0.0109
B	0.0049	0.0101	0.0017	0.0067
V	0.0136	0.0099	0.0078	0.0077
W	0.0147	0.0082	0.0080	0.0030
.	0.0073	0.0076	0.0075	0.0087
Y	0.0050	0.0058	0.0074	0.0026
,	0.0062	0.0001	0.0036	0.0060
)	0.0016	0.0039	0.0024	0.0024
(	0.0016	0.0039	0.0024	0.0024
-	0.0047	0.0034	0.0044	0.0053
1	0.0078	0.0024	0.0015	0.0054
K	0.0040	0.0023	0.0014	0.0008
/	0.0003	0.0016	0.0003	0.0019
J	0.0016	0.0014	0.0011	0.0000
X	0.0018	0.0013	0.0017	0.0008
2	0.0059	0.0013	0.0016	0.0031
+	0.0031	0.0011	0.0000	0.0000
=	0.0019	0.0010	0.0002	0.0000
Z	0.0002	0.0009	0.0011	0.0004
Q	0.0014	0.0008	0.0012	0.0004
3	0.0015	0.0007	0.0010	0.0018
0	0.0009	0.0005	0.0014	0.0050
-	0.0000	0.0005	0.0000	0.0000
"	0.0003	0.0003	0.0003	0.0000
4	0.0000	0.0003	0.0002	0.0001
`	0.0014	0.0002	0.0000	0.0000
:	0.0006	0.0002	0.0008	0.0004
8	0.0001	0.0002	0.0005	0.0015
@	0.0000	0.0002	0.0000	0.0000
5	0.0001	0.0002	0.0005	0.0014
9	0.0000	0.0002	0.0007	0.0034
*	0.0021	0.0002	0.0000	0.0000
;	0.0000	0.0002	0.0006	0.0000
6	0.0000	0.0001	0.0002	0.0015
.	0.0000	0.0001	0.0001	0.0000
7	0.0000	0.0001	0.0001	0.0001
'	0.0000	0.0001	0.0002	0.0004
<	0.0000	0.0000	0.0002	0.0000
!	0.0000	0.0000	0.0001	0.0000
#	0.0018	0.0000	0.0000	0.0000
$	0.0000	0.0000	0.0001	0.0000

NOTE: " " denotes space

used, six bits would be required; if a five-bit code is used instead, and one of the 32 code words is used as a shift character, 31 information characters can be transmitted using five bits and the remaining 25 information characters can be transmitted by using the five-bit code word reserved for a shift character followed by a five bit code word.

We model the Military Baudot code in terms of a code using a single one-character shift character. The Military Baudot code should perform somewhat better than predicted by the model because of the tendency for numbers and characters in the alphabet to occur sequentially and the actual implementation of the shift as a toggle operation.

More than one shift character can be used and these can be used sequentially to provide a whole family of different Baudot-like codes, which we call generalized Baudot codes. A generalized Baudot code is specified by its basic code length and the number of characters used as shift characters for each multiple of the block length. The generalized Baudot codes of most interest for Navy messages use code lengths which are multiples of 3, 4, or 5 (2 allows too few code words to build upon and 6 will block encode 58 characters).

The bits-per-character for the best single shift code is obtained as 5 + 5P, where P is the probability of occurrence of any of the 27 least commonly occurring characters. The average number of bits-per-character turns out to be 5.08, 5.06, 5.06, and 5.06, for narrative files I, II, III, and IV, respectively. For this code, one character is decoded in error per bit error and on the average 1.01 to 1.02 printed characters are in error per bit error, depending on the training file.

Consider a generalized Baudot code using more than one shift symbol. Suppose, in particular, that the shifts were used to produce a code with 15 words of length 4, 15 of length 8, 15 of length 12, and 13 of length 16. One of the first 16 code words is a shift, i.e., leads to a different interpretation of the next code word; one of these code words is reserved to lead to still another interpretation of the next code word; and one of these is reserved to lead to still another interpretation of the next code word. The average number of bits-per-character required to transmit information using this code is 4.68, 4.52, 4.54, and 4.52, for narrative files I, II, III, and IV, respectively. For this code, one character is decoded in error per bit error and on the average 1.13 to 1.17 printed characters are in error per bit error, depending on the training file.

A Baudot code based on length three code words provided about the same compression as one of length four, at the cost of greatly increased complexity.

HUFFMAN CODES

Using only the probabilities of a set of characters being transmitted, Huffman provided an organized technique for constructing efficient codes, i.e., using a minimum number of bits (on the average) to transmit characters. The procedure for constructing a Huffman code is illustrated in the following example drawn from reference 1.

Suppose that five characters, a, b, c, d, and e, with probabilities of occurrence 0.125, 0.0625, 0.25, 0.0625, and 0.5, respectively, are to be encoded (see figure 1).

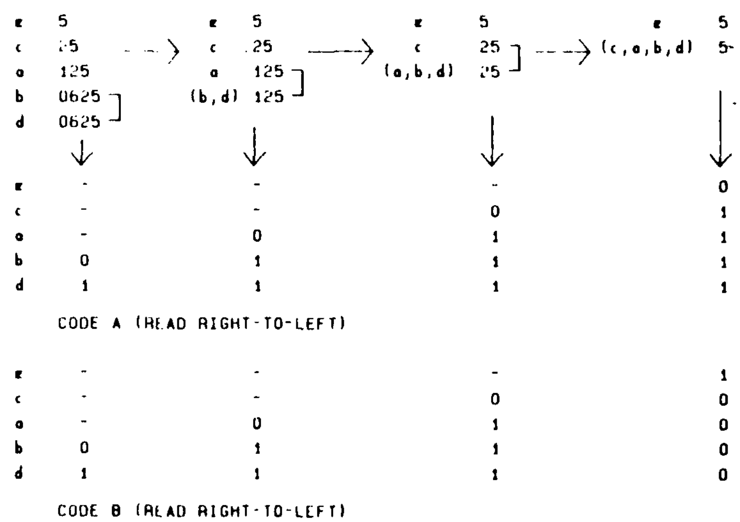


Figure 1. Two Examples of Huffman Coding

For this example, the Huffman procedure involves three regroupings of five characters. Grouped characters are indicated by (b,d), (a,b,d), and (c,a,b,d) along the top of figure 1. At each stage in this step, the two characters or group of characters with the lowest probabilities are grouped and the group is assigned the probability obtained by summing the probabilities of its members. The Huffman code is constructed based on the character groups by proceeding from right to left. Two of the many possible codes which can be assigned to the original character set by the Huffman construction process are illustrated in figure 1.

We discuss the construction of Code A first. Step 1: assign "0" to the most likely character "e" and "1" to the character set (c,a,b,d). These bits are the first bit in the code words assigned to the characters. The character "e" is distinguished from the characters "c", "a", "b", and "d" by the fact that the code for "e" begins with "0" and the others begin with "1". Step 2: no bit is assigned to "e", and a second bit is assigned to the remaining characters. This bit is chosen to distinguish "c" from "a", "b", and "d"; "0" is assigned to "c" and "1" is assigned to the other characters. Step 3: no additional bits are assigned to "e" and "c"; additional bits are assigned to distinguish "a" from "b" and "d". Step 4: no bits are assigned to "e", "c", and "a"; bits are assigned to distinguish "b" and "d".

Code B, also shown in figure 1, differs from code A in that at step 1, the character "e" is assigned "1" and the characters "c", "a", "b", and "d" begin with "0". The remaining steps are the same. Note that "0" and "1" can be assigned in either way at each step, leading to the construction of 16 different codes for the example shown in figure 1.

The example in figure 1 is very regular in that no reordering is necessary during the grouping of characters at the different stages of the construction process. This is not always the case. It is also worthwhile to note that the Huffman coding procedure can lead to block coding when all of the character probabilities are the same. For example, consider the case of eight characters: a,b,c,d,e,f,g, and h, each having a probability of 0.125. The first step leads to grouping g and h, the next step to grouping e and f, the next to grouping c and d, and the fourth step to grouping a and b. Each group is assigned a probability of 0.25. The next two steps lead to grouping e,f,g, and h, and to grouping a,b,c, and d. Each of these groups is assigned a probability of 0.5. It is easy to see that in this case each character is assigned a three-bit code word. In general, the Huffman code construction process for characters with differing probabilities of occurrence leads to a code with some characters having code words of the same length and other characters having code words of differing lengths.

Figure 2 shows the impact of introducing a single bit error into the code word assigned "a" for Code A and Code B. For Huffman codes, and other variable length codes, the impact of an error depends on the characters following "a". In the example, "abcde" is being transmitted. The impact of the single bit error is enclosed by brackets and an error count shown to the right for each of the two Huffman codes. For code A, an error in the first bit of the code word for "a" leads to it being incorrectly decoded into the two characters "e" and "c"; i.e., one character decoded in error and two characters printed in error. For code B, an error in the first bit of the code word for "a" leads to the next three characters being decoded in error for a total of 10 characters printed in error. For code A, the bit error does not lead to loss of character synchronization; while for code B, it does.

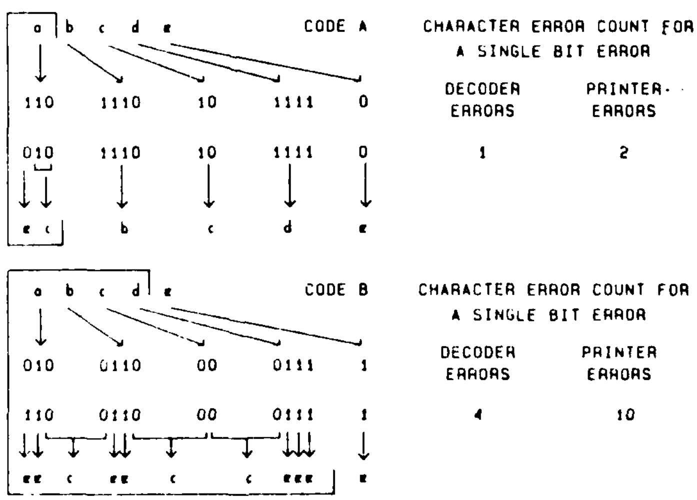


Figure 2. Examples of Error Propagation for Two Huffman Codes Providing the Same Compression

We turn now to the analysis of Huffman encoding of the four narrative files previously described. The structure of a Huffman code in the sense of its distribution of lengths of code words is determined by the probabilities of occurrence of the 58 characters in the encoded narrative file (provided some convention to treat equi-probable sets in the construction process is adopted).

Table 2 summarizes the code words assigned by the particular computer implementation of the Huffman constructed process that we used in our study. (It also contains a column of word lengths for a comma-free code. This column will be discussed later.) The code word lengths for Huffman encoding were obtained using the probabilities of occurrence of the characters presented in table 1 for the four narrative files. The order of the characters is the same in table 2 as that in table 1 and the characters are partitioned into sets of 15 characters to facilitate discussion.

The probability of occurrence of any of the first 15 characters listed in table 2 exceeds .84 for all the narrative files. The word lengths assigned to the first 15 characters based on the probabilities of occurrence of the characters in the different narrative files never differ by more than one bit. The word lengths are nearly the same for the next 15 characters and tend to differ greatly only for the least probable characters. Some of the differences between word lengths presented in table 2 for low probability of occurrence characters could have been lessened by adopting a different convention for equi-probability character sets than that used in our simulations. Nevertheless, use of any of the four narrative files as a training file should lead to similar compression results for Huffman (and later, comma-free) encoding of the narrative files.

Huffman code data compression performance is summarized by the average number of bits-per-character required to transmit the narrative files depending on the particular file (the training file) whose probabilities of character occurrence were used to construct the code. Table 3 summarizes the results of the Huffman code average bits-per-character calculations. Note that using narrative files II and III as training files gave nearly the same results. The maximum difference between two entries of the tables occurred when narrative file IV was used as a training file for narrative file I; however, the difference was only .23 bits-per-character.

TABLE 2. HUFFMAN AND COMMA-FREE CODE WORD LENGTHS FOR FOUR NARRATIVE FILES

CHAR	WORD LENGTHS FOR NARRATIVE FILE I	II	III	IV	COMMA-FREE	CHAR	WORD LENGTHS FOR NARRATIVE FILE I	II	III	IV	COMMA-FREE
" "	2	2	2	1	2	J	9	9	10	22	9
E	3	3	4	4	3	X	9	9	9	11	9
T	4	4	4	4	3	2	10	8	10	9	9
N	4	4	4	5	4	+	10	8	18	15	9
O	4	5	4	5	4	=	10	9	13	21	9
I	4	4	4	5	4	Z	10	12	10	12	9
A	4	4	4	5	5	Q	10	10	10	12	10
R	5	5	4	5	5	3	10	9	10	10	10
S	5	5	5	5	5	0	11	10	9	8	10
H	5	5	5	6	5	-	11	25	19	27	10
C	5	6	5	6	6	"	11	12	11	18	10
L	6	5	5	6	6	4	12	15	12	13	10
D	6	6	5	6	6	`	12	9	19	27	10
U	6	6	6	6	6	:	12	11	10	11	10
P	6	6	6	6	6	8	12	14	11	10	10
M	6	6	6	6	7	@	12	18	16	24	11
F	6	6	6	6	7	5	12	12	11	10	11
G	6	6	6	7	7	9	13	22	10	9	11
B	7	8	6	8	7	*	12	9	17	26	11
V	7	6	7	7	7	;	12	24	11	19	11
W	7	6	7	9	7	6	13	19	13	10	11
.	7	7	7	7	8	>	13	17	13	23	11
Y	7	8	7	9	8	7	14	20	13	14	11
,	7	7	8	8	8	'	15	21	12	11	11
)	8	7	9	9	8	<	16	25	12	20	11
(	8	7	9	9	8	!	17	23	13	17	12
-	8	8	8	8	8	#	18	13	15	25	12
1	9	7	9	8	8	$	18	16	14	16	12
K	9	8	9	11	9						
/	9	11	11	9	9						

TABLE 3. AVERAGE NUMBER OF BITS-PER-CHARACTER FOR HUFFMAN CODES AND DIFFERENT TRAINING FILES

TRAINING FILE	NARRATIVE FILE I	NARRATIVE FILE II	NARRATIVE FILE III	NARRATIVE FILE IV
I	4.04	4.05	4.24	3.77
II	4.12	3.95	4.19	3.73
III	4.12	3.96	4.15	3.72
IV	4.27	4.05	4.32	3.63

The entries in table 3 required to encode the training files are calculated directly as the sum of the probabilities of occurrence of a character with the length of the code word assigned to it by the Huffman construction process. The remaining table entries are obtained by multiplying each character probability of occurrence in the narrative file under consideration by the length of the Huffman code word assigned to that character and summing the results.

A computer program was written to search among the possible Huffman codes, which give the bit-per-character values presented in table 3, for the one performing best in a noisy channel. The original program reads a text file and counts the number of occurrences of each character; from this, a Huffman code is constructed using the construction process first described earlier. In order to search for a good code the specific choices made in the Huffman construction process were randomized. The probabilities of occurrence for each character and the assigned Huffman code words for each character were written to files so that the particular codes could be recovered if desired. Trials were run using the probabilities of occurrence of the characters in each of the four narrative files. The simulations were run by randomly introducing bit errors at a rate of 3 per 1000 bits. No attempt was made to model the impact of burst errors on the channel. It was felt that should burst errors pose a problem in the implementation of a particular code, it would always be possible to superimpose interleaving after data compression encoding and deinterleaving prior to data compression decoding.

We found that, regardless of the narrative file used as a training file, the poorest error performance results were obtained when processing narrative file IV (the most compressible) and the best error performance results were obtained for narrative file III (the least compressible). Figure 3 illustrates the dependency of the results upon the code selection process by presenting the average numbers of characters-decoded-in-error-per-bit-error for experiments run using narrative file II as a training file.

A best and worst code for each narrative file as a training file was selected for further analysis. Figure 4 presents the distributions of lengths of successive printed characters in error obtained for the best and worst of the eight cases analyzed. Note that any of the printed character error sequences may involve more than one bit error. However, the likelihood of two bit-error induced error sequences merging is very small for a bit error rate of 3 in 1000 and can be neglected; therefore, printed output character performance is summarized in terms of the average number of printed character errors per bit error, which we estimated by dividing the total number of character errors by the total number of bit errors introduced during a simulation run.

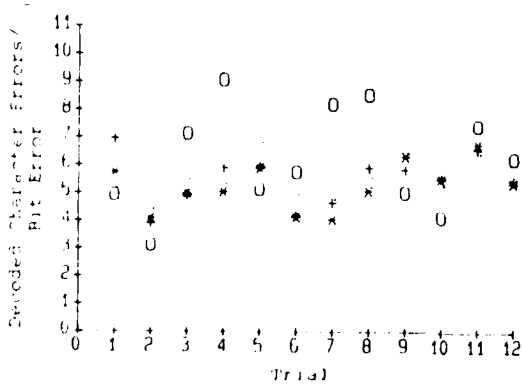


Figure 3. Huffman Code Error Propagation Results with Narrative II as Training File (. = Narrative I, + - Narrative II, * - Narrative III, o - Narrative IV)

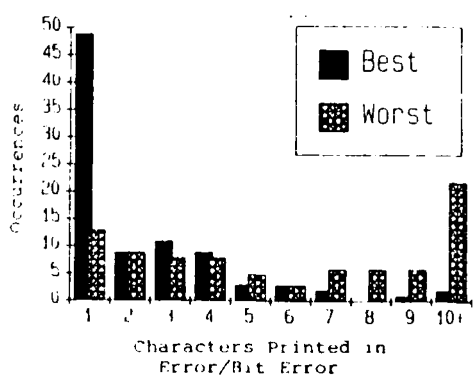


Figure 4. Distributions of Printed Characters in Error per Bit Error for Best and Worst Huffman Codes

There is a dramatic difference in the structure of the distributions for each of the narrative files used as a training file for the Huffman codes found to give the best and worst performance in a channel with errors. The best distributions have a preponderance of short length sequences (lengths one, two, and three) while the worst distributions tend to be relatively flat with the occurrence of extremely long character error sequences (35, 104, 65, and 90 for narratives I, II, III, and IV, respectively). The simulation for narrative IV shown in figure 4 only resulted in three printed character sequences longer than 7 characters, one each of 9, 10, and 14 characters. The average length of a printed sequence of character errors for this Huffman code, presented in table 4, was 2.4 printed character errors.

Some experiments were run using an operator-interactive program to determine the percentage of errors introduced into a text file through Huffman decoding of bit errors that could be corrected through narrative context. It appears possible to change a bit likely to be in error and then to use a standard spell check program to check whether reinitialization of Huffman decoding by the change leads to more reasonable results. The potential of such an algorithm could be assessed by using an operator-interactive program--with the operator choosing the decoding which provided text which made the most sense.

A 4271 character narrative file consisting of 88 lines and 4456 bytes was chosen to assess operator-interactive correcting of narrative character errors. Bit errors were introduced randomly at a rate of .005. This error rate would lead to an estimated 90 characters containing a bit error ((.005)(4271 characters)(4 bits/character)). These 90 bit errors led to 362 character decoding errors. After the interactive session the operator was able to reduce the number of character decoding errors to 85 errors (that is the number of character errors were reduced by 76 percent).

TABLE 4. HUFFMAN CODE WORDS FOR NARRATIVE FILE IV WHICH PROVIDED THE BEST PERFORMANCE IN A CHANNEL WITH ERRORS

CHAR	CODE WORD	CHAR	CODE WORD
" "	01	@	100010100010
!	10001010000000011	A	1110
"	1111110101	B	0001101
#	10001010000000100	C	11110
%	10001010000000101	D	000011
'	100010100000001	E	110
(	00100001	F	100011
)	00100000	G	111110
*	11111101001	H	10000
+	0010101000	I	1010
,	1111110	J	111111110
-	10001011	K	001010101
.	0010001	L	000010
/	111111111	M	001001
0	00101011101	N	1001
1	001010110	O	1011
2	0010101111	P	001011
3	1111111011	Q	1000101001
4	001010111000	R	00000
5	100010100011	S	00010
6	0010101110010	T	0011
7	10001010000001	U	000111
8	100010100001	V	0001100
9	0010101110011	W	0010100
:	100010101100	X	111111100
;	11111101000	Y	1000100
<	1000101000000000	Z	1000101010
=	0010101001	^	100010101101
>	1000101000001	~	10001010111

COMMA-FREE CODES

Comma-free codes are binary codes so constructed that it is possible to identify individual code words prior to decoding the received bit stream. We analyze a family of comma-free codes, known as "suffix/prefix" codes, found by R. A. Scholtz [reference 2].

In order to illustrate the ideas involved in Scholtz's construction process, we discuss a particularly simple example of the Scholtz construction process. Scholtz constructs sets of code words sequentially. We begin with the set of two code words (0,1). The next code set is obtained from this set by choosing "1" as a suffix. This set consists of (0,01,011,0111,01111,...). A next set is obtained by choosing an element of this set as either a suffix or a prefix. If "0" is chosen as a prefix, the code set becomes (0...01...1 with at least one "0" and one "1") (we call this code the suffix-prefix comma-free code); if "0" is chosen as a suffix, the code set becomes (01...10...0 with at least one "1") (we call this code the suffix-suffix comma-free code); if "01" is chosen as a suffix, the code set becomes (0...01...101...01 with zero or two or more "1"s). Generally speaking, new code words can be constructed by either using suffixes or prefixes. The process can be carried out any number of times.

Figure 5 shows the operation of the three-step process used to insert "commas", which corresponds to the two-step construction process used to construct the suffix-suffix code described in the last paragraph. (Individual code words are enclosed in brackets and the bits maintained in alignment in figure 5 to aid the reader. The transmitted and received bit stream would consist simply of the bits enclosed in these brackets with no indication of where one code word ended and another began.) The comma insertion process parallels the code construction process and the reader could readily verify that it reconstructs the correct code words in the absence of errors, except possibly at the beginning or end of the decoded sequence. It proceeds by first inserting commas between all the bits and then successively deleting those according to rules based on the suffix choices. For example, corresponding to choosing "1" as a suffix, commas are removed preceding "1"s in the second step of the comma insertion process. Other algorithms are available to insert commas for some of the comma-free codes. For example, for a suffix "1" prefix "0" code, one need only insert a comma between every string of "1"s and string of "0"s to identify the code words.

We turn now to the selection of the comma-free code which would provide the best data compression for a narrative file with particular character probabilities of occurrence. R. A. Scholtz does not discuss how to match his comma-free code construction process to the probabilities of occurrence of the characters to be encoded to provide the best compression. We found how to survey the possible codes in terms of the distributions of their code word lengths. The survey can be conducted without specifying the particular code word chosen at each step of the construction process, or whether the chosen word at each step is used as a suffix or a prefix. All that need be specified is the lengths of the words chosen for suffixes and prefixes.

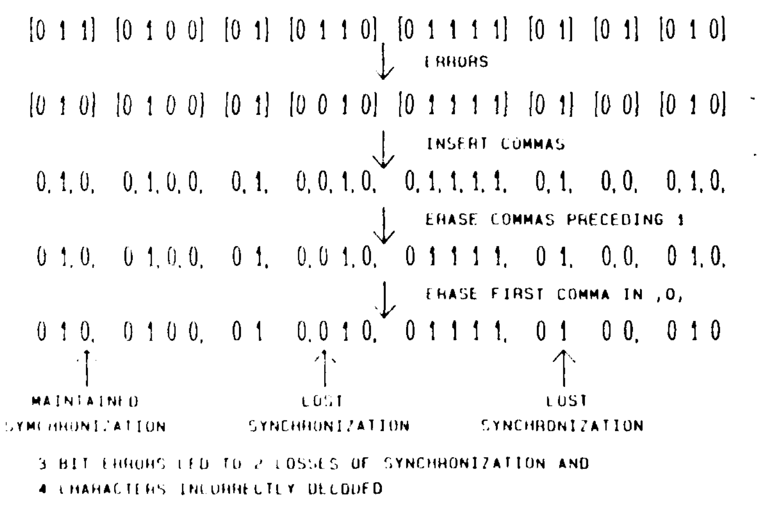


Figure 5 An Example of the Comma Free Algorithm to Insert "Commas" in a Channel with Errors

A natural way to survey the codes is to survey them inductively based on the construction steps. Toward this end, let C[k] denote the set of code words produced after the first k steps of the construction process; let C[1] = {0,1} be the starting point in the construction process; and let n[k](j) denote the number of code words of length j in set C[k]. In the suffix/prefix construction process, a word used as a suffix or prefix can no longer be used as a code word. To take this into account let n'[k](j) = n[k](j) - 1, if j = s, and = n[k](j), if j ≠ s, where s is the length of the suffix or prefix chosen to construct the (k+1)-th code set from the k-th code set.

The inductive formula for the number of comma-free code words of length j resulting from the choice of a suffix or prefix of length s in the (k+1)-th construction step is given by n[k+1](j) = n'[k](j) + n'[k](j-s) + ... + n'[k](j-ns), with the convention that n'[k](j-ns) = 0 if j - ns < 1.

The above formula allows easy compilation of tabular summaries of distributions of code word lengths for available comma-free codes constructed using the suffix/prefix process. Table 5 illustrates its use. With the exception of the first column, the numbers of code words in a code are only summarized up through the length of code word needed to allow the coding of 58 characters. The codes summarized in table 5 begin with C[1] = {0,1} and each new code set C[2] through C[6] is obtained from the previous one by using one of shortest available code words as a suffix or prefix in the construction process.

We turn now to the selection of a comma-free code to encode the narrative files used earlier to assess Huffman codes. The probabilities of occurrence for the character set used for our 58 character simulations of the Huffman code, has the property that the character probabilities fall off rapidly from the most used characters to the least used characters (as shown in table 1 which was presented earlier). In such a situation, if we could closely match the code word lengths provided by the Huffman code constructed for the given character probabilities of occurrence for the first 10 to 15 characters, we would expect very similar compression performance from that comma-free and a Huffman code.

Table 2 (previously discussed) shows how closely the simplest suffix/prefix candidate code word lengths match those provided by the Huffman codes. The first four columns present Huffman word lengths and a fifth column presents the word lengths for any of the suffix-prefix comma-free codes obtained by use of "0" and "1" as suffixes or prefixes in a two-step construction. The assignment of code words to characters is optimum for narrative file II. However, as can be seen from table 2, this assignment leads to excellent word length agreement through the first 40 characters for all of the narrative files so no other word assignments were studied.

Table 6 presents a comparison between Huffman code bits-per-character values and comma-free code bits-per-character values for the code word to character assignments shown in table 2. The next best comma-free code appears to be the code (1,1,3) for which similar calculations revealed a penalty of .114 (rounded down to three significant places) bits-per-character for using this comma-free code instead of the Huffman code for narrative file II.

There are choices in the construction of comma-free codes leading to the same distribution of code word lengths. The behavior of the code in a noisy channel depends on these choices. We single out the suffix-prefix and suffix-suffix codes for detailed study. These codes represent the two fundamentally different codes providing the compression summarized in table 6.

TABLE 5. PARTIAL SURVEY OF THE DISTRIBUTIONS OF CODE WORD LENGTHS FOR COMMA-FREE CODES

WORD LENGTH	COMMA-FREE CODE C[1]	C[2]	C[3]	C[4]	C[5]	C[6]
1	2	1				
2		1	1			
3		1	2	2	1	
4		1	3	3	3	3
5		1	4	6	6	6
6		1	5	8	9	9
7		1	6	12	15	18
8		1	7	15	21	27
9		1	8	18	27	
10		.	9	24		
11		.	10			
12		.	11			

TABLE 6. COMPARISON OF BITS-PER-CHARACTER VALUES OF HUFFMAN AND COMMA-FREE CODES

NARRATIVE FILE	AVERAGE NUMBER OF BITS-PER-CHARACTER	
	HUFFMAN CODE	COMMA-FREE CODE
I	4.04	4.10
II	3.95	4.00
III	4.15	4.26
IV	3.63	3.81

The impact of bit errors on character errors can be determined analytically for the suffix-prefix and the suffix-suffix codes. The basic observation is that for these two codes a bit error in the middle word of three code words w[1]w[2]w[3] always leads to a bit sequence which can be expressed as three other code words w'[1]w'[2]w'[3] with at most two of the code words in error. The probability that a particular bit in w[2] is in error is given by the probability that the character w[2] represents occurs times the probability that bit in w[2] is in error, which is just 1 divided by the length of w[2].

Generally speaking, four probabilities determine the error propagation properties of a comma-free code: P[D], the probability that a bit error leads to the deletion of the comma separating two code words; P[M], the probability that a bit error leads to the misplacement of a comma; P[A], the probability that a bit error leads to the addition of a comma; and P[N], the probability that a bit error leads to no change in the placement of the commas. The impact of a bit error on the comma determines the number of characters decoded in error and the number of characters output by the decoder in error. In particular: (1) if a bit error leads to comma deletion then two characters are incorrectly decoded as a single character or not decodable; (2) it a bit error leads to comma movement then two characters are incorrectly decoded into two characters; (3) if a bit error leads to the insertion of a comma (always within the code word with the bit error) then one character is incorrectly decoded into two characters; and (4) if there is no change in the commas then one character is incorrectly decoded into a single character.

The required calculations for a particular assignment of the suffix-prefix code words to a 58-character set are easy but tedious. We omit the majority of the details (see reference 4 for them) and summarize the results of the calculations. Calculations were carried out for the probabilities of occurrence of the characters in narrative file II. Similar results are expected for the remaining three narrative files.

For the assignment of suffix-prefix codes to characters described above and summarized by table 2, the following statistics were obtained: P[D] = .42, P[M] = .21, P[A] = .16, and P(N) = .21. Note that about three-quarters of the contribution to P[D] is that provided by the code word "01" assigned to the space character with probability of occurrence 0.317. The average number of characters decoded in error per bit error is (.42)(2) + (.21)(2) + (.16)(1) + (.21)(1) = 1.63. The average number of incorrect printed characters per bit error is given by (.42)(1) + (.21)(2) + (.16)(2) + (.21)(1) = 1.37. Note that these values are obtained by treating words too long to be decoded because they exceed the longest word assigned one of the 58 characters as being incorrectly decoded. (Such characters could be decoded into a 59th character indicating an error has occurred.) The calculations presented clearly indicate that the performance of the suffix-prefix code in an error channel is considerably better than the performance of any Huffman code that we found.

A two-step comma-free code construction using a one-bit prefix and a one-bit suffix, no matter what choices are made, leads to code words either of the form 0...01...1 or 1...10...0 with each code word containing at least one "1" and one "0". One of these codes can be obtained from the other by interchanging "1"s and "0"s. If this were done to the assignment of code words, the same probabilities would be obtained as for the code word assignment before the interchange. Thus all the prefix-suffix codes using a one-bit prefix and a one-bit suffix would for these assignments have the same error statistics.

We turn now to estimating the impact of errors on the suffix-suffix code discussed earlier. Recall that the code words for this code have the structure 01...10...0 with at least one "1". This code differs from the suffix-prefix code in that (1) the impact of an error in the first bit position of a code word depends on the ending of the previous code word and (2) the impact of an error in the second bit position depends on whether or not the second bit is the only "1" in the code word. Again calculations were only carried out for the probabilities of occurrence of the characters in narrative II.

An error in the first bit leads to the movement of a comma or the deletion of a comma depending on whether the first code word ends in "0" (probability of 0.51) or ends in "1" (probability of 0.49). It follows that the probability that an error in the first bit leads to the movement of a comma is .158 and the probability that an error in the first bit leads to the deletion of a comma is .152.

An error in the second bit leads to the deletion of a comma or the addition of a comma, depending on whether or not the second bit in error was the only "1" in the code word. It turns out that the probability that an error in the second bit will lead to the deletion of a comma is 0.217 (rounded to three places) and that the probability that an error in the second bit leads to the addition of a comma is 0.95.

The remaining calculations are similar to those for the suffix-prefix case. We found that the probability that a bit error leads to the addition of a comma through changing other than the first or second bit is .18 and the probability that a bit error leads to no change in the commas through changing other than the first or second bit is .20.

From these calculations, it follows that P[D] = .37, P[M] = .16, P[A] = .27, and P[N] = .20 so that the average number of characters decoded in error per bit error is 1.51 and the average number of printed characters which are incorrect per bit error is 1.45. These statistics can be seen to apply to all suffix-suffix and prefix-prefix codes with length one suffixes or prefixes.

The result that a single bit error leads to at most two character errors (decoded or printed), established for the two simplest kinds of comma-free codes, can be extended to other comma-free codes. Some additional terminology is needed to facilitate the discussion of general comma-free codes. Let k denote the kernel of the code under construction, p(i), i = 1, 2, ... denote the prefixes used in the code under construction, and s(j), j = 1, 2, ... denote the suffixes used in the code under construction. Suppose that the codes under discussion satisfy: (1) k = "0" or "1", (2) both "0" and "1" are used as either prefixes or suffixes, and (3) the length of the prefix or suffix used in k-th construction step is less than or equal to the length of the prefix or suffix used in the (k+1)-th construction step. A code is called exhaustive if for each of the steps in the code construction process the code word chosen as either a prefix or suffix is one of the shortest code words available.

For an exhaustive comma-free code, a single bit error can lead to at most two characters decoded in error. To establish this result, consider (1) an incoming sequence of bits as a sequence of kernels, prefixes, and suffixes, and (2) the comma-insertion algorithm (after the first step) consists of deleting commas between the kernels, prefixes, and suffixes. Now, let us discuss the potential impact of a single bit error occurring in a kernel or in a prefix or suffix of the code words.

Let us denote the word with a bit error by use of "^". Consider the incoming sequence of binary bits parsed into codewords w(1)w(2)w^(3)w(4)w(5). Under what conditions will the comma separating w(1) and w(2) or the comma between w(4) and w(5) be altered as a result of a bit error somewhere in the codeword w(3)? Each of these words is constructed from the kernel and prefixes and suffixes, as described above so that for the comma between w(1) and w(2) to be erased by the comma-insertion algorithm, the prefix or kernel beginning w(2) must be transformed into a suffix through a bit error in w(3). Since none of the bits in w(2) are in error, this can only happen if the addition of bits to the bits of w(2) has created a suffix used in the construction process; i.e., there exists a code word of shorter length in the code than some suffix in the code, a contradiction. For the comma between w(4) and w(5) to be erased by the comma-insertion algorithm, the suffix or kernel ending w(4) must be transformed into a prefix through a bit error in w(4). Since none of the bits in w(4) are in error, this can only happen if the addition of bits to the bits of w(2) has created a prefix used in the construction process, a contradiction.

It is clear that one could improve upon the results by examining the non-exhaustive codes to see if either of the above phenomena can occur for a particular selection of prefixes or suffixes. A cursory examination allowed us to establish that for the codes with sequences of suffixes or prefixes with the lengths indicated by (1,1,3), (1,1,3,3), (1,1,2,4), and (1,1,4), a single bit error can never lead to four or more character decoding errors.

It is also possible to use Huffman or comma-free codes to encode characters based on one or more of the previous characters encoded. We refer to this as encoding based on conditional probability of occurrence of characters. Since the error propagation properties of Huffman codes were so much worse than those for comma-free codes, we restricted our attention to comma-free encoding of characters conditioned on the occurrence of previous characters.

The structure of the comma-free codes limits the impact of bit errors. However, given an error has occurred in an encoded character, then it will be decoded in error, the next character will be decoded in error if conditioned on it, the next code word decoded in error if conditioned on either of the previous characters, and so on. To prevent decoding errors from propagating in this manner, it is necessary to reinitialize the coding process fairly often.

Military and commercial messages are transmitted using characters of various types, categorized as the set of letters { A,B,C, ..., X,Y,Z }, the set of numbers { 0, 1, 2, ..., 9 }, and the set of symbols { punctuation symbols, special characters, control characters }. The basis of the conditional probability encoding approach is to reinitialize the encoding process whenever c is a symbol. Furthermore, in order to limit the propagation of decoding errors, symbols are encoded independent of previously encoded characters.

Off-line processing would be used to determine the assignment of comma-free code words to characters. The overall probabilities of occurrence can be used to assign code words to symbols (although we found that this was not optimum). The crucial assignment is the code word assigned to the space. (We found that the assignment of a three-bit code word to the space gave the best compression.) The variable length code words assigned to fixed length words representing characters or numbers are the remaining code words.

We assign a code word to a letter depending on whether or not it is the first letter of a word and to a number depending on whether or not it is the first number in a sequence of numbers. The comma-free code words assigned unconditionally to letters or numbers are based on the probabilities that a character occurs as the first letter in a word (including one letter words) or the first number in a sequence of numbers (including single numbers); the conditional assignment of comma-free code words to letters and numbers is based on the probabilities that a character follows a specific letter or follows a specific number.

The data compression provided by the comma-free encoding approach described above was estimated by the following formula: $3/(L+1) + N[\text{Uncond}]/(L+1) + N[\text{Cond}](L-1)/(L+1)$, where $N[\text{Uncond}]$ = the average number of bits assigned the starting letter in a word, $N[\text{Cond}]$ = the average number of bits encoding a character conditioned on the receipt of a previous character, and L = the average length of a word (with symbols treated as length 1 words). This formula neglects the contributions of numbers conditioned on other letters or numbers, which are extremely rare in the narratives. The first term, $3/(L+1)$, would be accurate if blanks separated all the words, another fairly valid assumption because of the absence of short sentences in the file manuscript and the usual practice of separating them from letters and numbers using spaces. Table 7 summarizes the results obtained using the above formula. By using a length 3 code word for the blank, and making effective use of the length 2 code word for the most commonly occurring character starting a word, the table shows that the bits to encode spaces and beginning of words are about the same as the bits to encode the remainder of words.

We estimate the average number of characters decoded in error for the suffix-prefix comma-free encoding of characters conditioned on the previous character for narrative file II by considering the impact of bit errors on an average word followed by a space. The expected structure of a five-character word followed by a space is (4.4 bits)(3.5 bits)(3.5 bits)(3.5 bits)(3.5 bits)(3 bits), where we have rounded 3.55 down to 3.5 bits to compensate for rounding 4.69 up to 5 letters. Thus, a word is expected to consist of 18.4 bits and a word followed by a space of 21.4 bits. We proceed by estimating the number of characters in error as a result of bit errors in the different bit positions. A bit error in the first bit of a code word of a letter leads to the previous character being decoded in error; such errors contribute $(1/21.4)(6+5+4+3+2) = .93$ characters in error. A bit error in any of the other bits of the letter code words, with the exception of the very last bit of the code word of the last letter in the word, leads to the rest of the word being decoded in error; these bit errors contribute $(1/21.4)[(3.4)5+(2.5)4+(2.5)3+(2.5)2+(1.5)] = 1.92$ characters in error. If the last bit of the code word of the last letter of the word, or the first bit of the space is in error, both the last letter and the space will be decoded in error, and therefore the following word will be decoded in error; these bit errors contribute $(2/21.4)(7) = .65$ characters in error. Bit errors in the remaining two bit positions of the space lead to it and the following word being decoded wrong, so they contribute $(2/21.4)(6) = .56$ characters in error. Totalling these contributions leads to an estimate of 4.1 characters in error per bit error.

Table 7 allows us to estimate the potential payoff of encoding the third through last letters of words based on the occurrence of two previous letters. Since there is a single remaining comma-free 2-bit code word, a single 3-bit code word, and the remaining code words are of length 4 or more, the least length that one might expect for the doubly conditioned encoded characters is $(1/2)2 + (1/4)3 + (1/4)4 = 2.75$. Then a lower bound on the expected overall average number of bits would be $(1/2)(3.6) + (1/2)(2.8) = 3.2$ for this encoding approach. We would expect about 3.3 to 3.4 bits per character performance if we carried out the calculations more exactly.

TABLE 7. AVERAGE NUMBER OF BITS AND WORD LENGTHS FOR COMMA-FREE CODES USING CONDITIONAL PROBABILITIES

NARRATIVE FILE	AVERAGE NUMBER OF BITS: START OF WORD	REMAINDER OF WORD	OVERALL	AVERAGE WORD LENGTH
I	4.51	3.55	3.62	4.64
II	4.44	3.55	3.61	4.69
III	4.71	3.64	3.71	5.15
IV	4.53	3.54	3.62	4.91

The estimated number of character errors per bit error for a doubly conditioned comma-free encoding scheme using a suffix-prefix code is the same as for a singly conditioned comma-free encoding scheme; all the letters or numbers after an error are in error until the encoding process is reinitialized whether singly or doubly conditioned.

SUMMARY AND CONCLUSIONS

The performance of Huffman codes, suffix/prefix comma-free codes, block codes, and some variants of Baudot codes was obtained for the encoding of narrative files of a personal computer for a 58-character set. Figure 6 summarizes the results of this investigation. The performance of each encoding approach is summarized by its compression performance measured by the average number-of-bits per character (the y-axis) and by its error propagation statistics measured by the average number of characters decoded in error per bit error (the x-axis). A word error was treated as equivalent to five character errors to allow comparison of character and word encoding approaches.

The Military Baudot code uses two shift keys ("LTRS" and "FIGS"), which we model mathematically as a single shift key generalized Baudot code, to reduce the number of bits required to transmit information from 6 to about 5.06. Generalizations of this construction can further reduce the average number of bits required to around 4.5 bits-per-character while maintaining a basic block structure.

A suffix/prefix comma-free code can be constructed which provides nearly the same data compression as a Huffman code, about 4 bits-per-character. For the character set and probabilities of occurrence of the characters of the set used in the Huffman simulation, the penalty varied from a low of .05 bit per character to a high of .18 bit per character for four narrative files. A single bit error can lead to at most two character errors for the above prefix/suffix codes while a single bit error was found to lead to as many as 90 character errors for a Huffman code. Relative to a 1-shift Baudot code, the best Huffman code requires an average 24% fewer bits for encoding with 43% more decoded character errors, while the best comma-free code requires 22% fewer bits for encoding with 18% more decoded character errors.

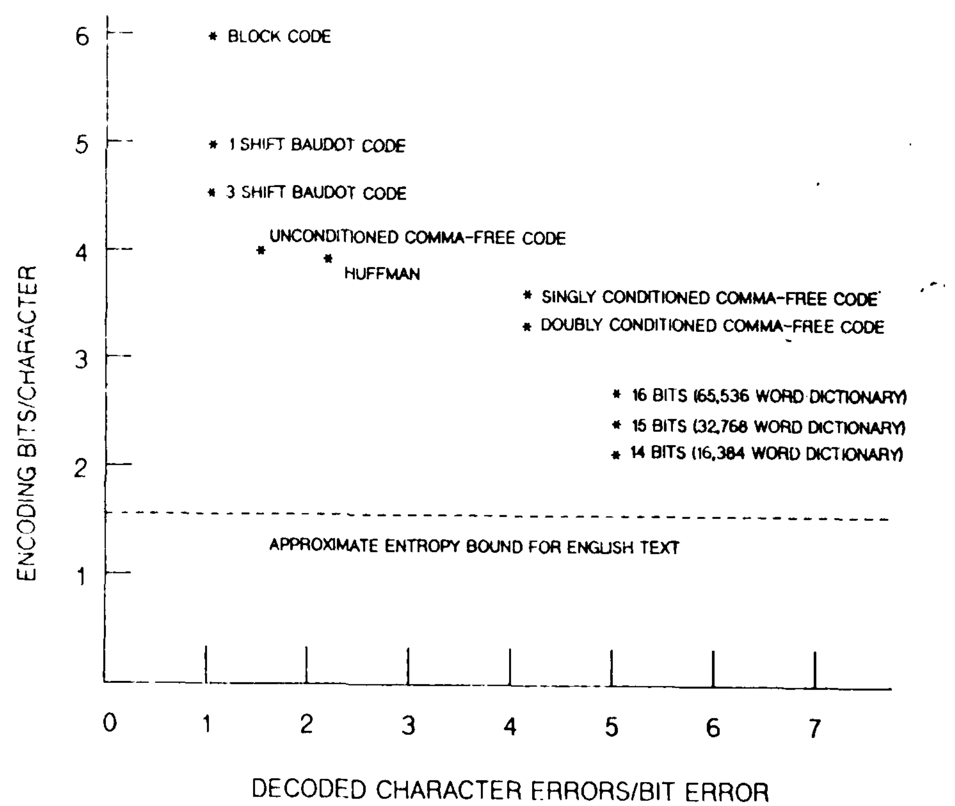


Figure 6. Data Compression and Error Propagation in Noisy Channels

The use of comma-free encoding for words with non-letters and non-numbers encoded independently of the previous character led to an estimated 3.6 bits per character along with 4.1 characters in error per bit error for a singly conditioned encoding scheme, and about 3.3 bits per character along with 4.3 characters in error per bit error for a character for a doubly conditioned encoding scheme. Relative to a 1-shift Baudot code, the singly conditioned scheme requires 29% fewer bits for encoding and the doubly conditioned scheme about 35% fewer bits both with 289% more decoded character errors.

Block encoding of words for dictionaries ranging in size from 16,384 to 65,536 words requires 2.33 to 2.67 bits per character with one word error per bit error.

The following conclusions were drawn as a result of the investigation:

(1) For unconditioned character encoding, comma-free codes significantly outperform Generalized Baudot codes and Huffman codes in a noisy channel. They provide nearly the same compression and have significantly fewer decoded or printed character errors than Huffman codes.

(2) Additional compression is achievable by the use of comma-free encoding of characters based on their conditional probability of occurrences.

LIST OF REFERENCES

1. Huffman, D., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers, Vol. 40, pp. 1098-1101, September 1952.

2. Scholtz, R., "Codes with Synchronization Capability", IEEE Transactions on Information Theory, Vol. IT-12, No. 2, April 1966.

3. Scholtz, R., "Maximal and Variable Word-Length Comma-Free Codes", IEEE Transactions on Information Theory, Vol. IT-15, No. 2, March 1969.

4. Bond, J., "The Performance of Data Compression Codes in Channels with Errors", personal correspondence of author, 19 February 1987.

5. Bond, J., "Variable Length Data Compression Encoder", Patent Disclosure, 1987. (available from author)

END
DATE
FILMED
6-88
DTIC